Inexact Quasi-Newton methods for sparse systems of nonlinear equations

Luca Bergamaschi [a], Igor Moret [b], and Giovanni Zilli [a]

[a] *Dipartimento di Metodi e Modelli Matematici per le Scienze Applicate*
Università di Padova, Via Belzoni 7, 35131 Padova, Italy
E-MAIL {*berga,zilli*}*@dmsa.unipd.it*

[b] *Dipartimento di Scienze Matematiche, Università di Trieste, Italy*
E-MAIL *moret@univ.trieste.it*

Abstract

In this paper we present the results obtained in solving consistent sparse systems of n nonlinear equations $F(x) = 0$, by a Quasi-Newton method combined with a p block iterative row-projection linear solver of Cimmino-type, $1 \leq p \ll n$. Under weak regularity conditions for F, it is proved that this Inexact Quasi-Newton method has a local, linear convergence in the energy norm induced by the preconditioned matrix HA, where A is an initial guess of the Jacobian matrix, and it may converge superlinearly too. The matrix $H = [A_1^+, \ldots, A_i^+, \ldots, A_p^+]$, where $A_i^+ = A_i^T(A_i A_i^T)^{-1}$ is the Moore-Penrose pseudo inverse of the $m_i \times n$ block, A_i is the preconditioner. A simple partitioning of the Jacobian matrix was used for solving a set of nonlinear test problems with sizes ranging from 1024 to 131072 on the CRAY T3E under the MPI environment.

Key words: Sparse nonlinear problems, Inexact Newton method, Quasi-Newton, row-projection method, parallel iterative solver.

1 Introduction

We are concerned with the numerical solution of a system of nonlinear equations

$$F(x) = 0 \qquad F = (f_1, ..., f_n)^T \tag{1}$$

where $F : R^n \to R^n$ is a nonlinear C^1 function, and its Jacobian matrix $J(x)$ is sparse and n is large. For solving (1) we present an iterative procedure which combines a Quasi-Newton method (see [8], chap. 8) with a row-projection (or

row-action) linear solver of Cimmino type [5], particularly suited for parallel computation (truncated or Inexact version of the Quasi-Newton method, in the sense of [7]). In the recent years there has been a growing of interest in the row projection methods for solving large sparse nonsymmetric linear systems [1,3,15,16]. Several numerical experiences have shown that they may be competitive with other iterative solvers, in particular when the Conjugate Gradient (CG) acceleration is employed. In [16] and [17] some preliminary numerical results in solving large sparse systems of nonlinear equations by an Inexact Newton-Cimmino and an Inexact Quasi-Newton Cimmino (see Section 2) methods are obtained (using a Transputer network and the Cray T3E).

Here below, referring to block Cimmino method, we give the general lines of this procedure.

Let $As = b$ be a nonsingular linear system of size n to be solved. Let us partition A into p row-blocks : $A_i, i = 1, \dots, p$, of size $n_i \times n$, i.e. $A^T = \left[A_1^T, A_2^T, \dots, A_p^T\right]$, and partition the vector b correspondingly. Then the original system is premultiplied (*preconditioning*) by

$$H = [A_1^+, \dots, A_i^+, \dots, A_p^+] \tag{2}$$

where $A_i^+ = A_i^T(A_i A_i^T)^{-1}$ is the Moore-Penrose pseudo inverse of A_i. We obtain the equivalent system $HAs = Hb$,

$$(P_1 + \cdots + P_p)s = \sum_{i=1}^{p} A_i^+ A_i s = \sum_{i=1}^{p} A_i^+ b_i = Hb, \tag{3}$$

where for each $i = 1, \dots, p$, $P_i = A_i^+ A_i$ is the orthogonal projection onto range(A_i^T). We solve this linear system by the iterative (Richardson) method, obtaining the row-block Cimmino method [15]

$$s_{m+1} = s_m + \omega \sum_{i=1}^{p} A_i^+ (b_i - A_i s_m) = s_m + \omega \sum_{i=1}^{p} \delta_{i,m}. \tag{4}$$

Clearly, as A is nonsingular, the matrix HA as sum of the orthogonal projectors $\sum_{i=1}^{p} P_i$ is symmetric and positive definite. Therefore we will use the Conjugate Gradient (CG) method to approximate the solution of (3). It is well known that, for every starting guess s_0, the CG method produces a sequence of approximations s_m to s such that

$$\|s - s_m\|_{HA} \leq c^m \|s - s_0\|_{HA}, \quad \text{for every } m = 1, 2, \dots, \tag{5}$$

for some $c < 1$, with $\|x\|_{HA} = [(HAx)^T x]^{1/2}$, for $x \in R^n$. Here below the notation $\|\cdot\|_{HA}$ will denote also the matrix norm induced by the vector norm $\|\cdot\|_{HA}$.

At every CG iteration the pseudoresiduals $\delta_{i,m}$ are evaluated without explicitly computing matrices A_i^+, and hence without inverting $A_i A_i^T$. They are computed by solving the following p linear (underdetermined) subproblems:

$$A_i \delta_{i,m} = (b_i - A_i s_m), \quad 1 \leq i \leq p$$

in the least squares sense. In this paper we solve them *concurrently* with the iterative algorithm LSQR [13]. Other choices are possible, like the augmented system approach used in [1]. Moreover, to simplify the solution of the least squares subproblems at each step of the inner iteration, a suitable block row partitioning of the matrix A may be adopted in such a way that $A_i A_i^T = I$, and consequently, $A_i^+ = A_i^T$. This partitioning [15] is always possible for every sparse matrix and produces a number p of blocks A_i whose rows are mutually orthogonal. We report in Section 5 some numerical results in this direction.

2 The Inexact Quasi-Newton method

Now let us turn to equation (1). Combining the classical Newton method [8] and the block Cimmino method of section 1 we obtain the block Inexact Newton-Cimmino algorithm [16,17], in which at a major outer iteration the linear system $J(x_k)s = -F(x_k)$, where $J(x_k)$ is the Jacobian matrix in x_k, is solved in *parallel* by the block Cimmino method.

The class of Inexact Newton methods [7] are a variant of the classical Newton method for solving a system of nonlinear equations. It is based on the approximate solution of the linearized system by an iterative method (Cimmino in our case) which is stopped at an accuracy related to the residual of the previous nonlinear iteration. Namely, instead of solving

$$J(x_k)s_k = -F(x_k)$$

exactly, the correction s_k satisfies

$$\|J(x_k)s_k + F(x_k)\| \leq \eta_k \|F(x_k)\|,$$

where $\{\eta_k\}$ is a forcing sequence. In [7] it is proved that the method has a local, linear convergence in an appropriate norm, and it may converge superlinearly or even quadratically under convenient assumptions.

ALGORITHM: INEXACT NEWTON-CIMMINO

Let x_0 be the initial vector, $k = 0$.
WHILE $\|F(x_k)\| > \varepsilon_1 \|F(x_0)\|$ DO
- compute $A = J(x_k)$

- partition $A^T = (A_1^T, \ldots, A_p^T)$, $\qquad F \equiv F(x_k)^T = (F_1^T, \ldots, F_p^T)$
- compute the preconditioner H
- solve $HAs_k = -HF$ by the Conjugate Gradient method noting that the product $y = HAv$ is implemented in parallel as

$$y = \sum_{i=1}^{p} A_i^T (A_i A_i^T)^{-1} A_i v_i$$

  where every term of the sum is computed by a different processor i.
- $x_{k+1} = x_k + s_k$
- $k = k + 1$

END WHILE

The exit test for the Cimmino iterations is $\|r_{k,m}\| \leq \varepsilon_2 \|F(x_k)\|$.

To overcome the expensive computation of the Jacobian matrix at every nonlinear iteration, the Quasi Newton method has been developed, which tries the solution of $F(x) = 0$ by computing at each Newton iteration a (secant) approximation B_k to $J(x_k)$. We now consider the following Broyden-like method:

ALGORITHM: INEXACT QUASI-NEWTON CIMMINO

Let x_0 be the initial vector, $k = 0$, $A \equiv J(x_0)$.
Let H be defined by (2). Set $B_0 = HA$.
WHILE $\|F(x_k)\| > \varepsilon_1 \, \|F(x_0)\|$ DO
Compute an approximation s_k to the solution of the linear system

$$B_k s = -HF(x_k) \tag{6}$$

Set

$$\begin{aligned} x_{k+1} &= x_k + s_k \\ y_k &= H\left(F(x_{k+1}) - F(x_k)\right), \\ u_k &= \frac{(y_k - B_k s_k)}{\|s_k\|_{HA}} \\ v_k &= \frac{HAs_k}{\|s_k\|_{HA}} \\ B_{k+1} &= B_k + u_k v_k^T, \qquad k = k + 1. \end{aligned}$$

END WHILE

Although, the fixed approximation A to the initial Jacobian $J(x_0)$ is updated in each step, we refer to the above process as a Quasi-Newton method. Then we also obtain an Inexact Quasi-Newton method since the correction at each step is determined by solving approximately the linearized equation (6). It is easy

to check that $B_{k+1}s_k = y_k$ *(secant equation)* and $\|B_{k+1} - B_k\|_{HA} \leq \|B - B_k\|_{HA}$ for any B for which $Bs_k = y_k$ *(least change condition).*

Concerning the solution of (6) we proceed as follows. Setting $z = HAs$ and $t_i = s_i/\|s_i\|_{HA}$ for every $i = 0, \ldots, k-1$, we rewrite (6) as

$$z + \sum_{j=0}^{k-1} u_j t_j^T z = -HF(x_k) \qquad k \geq 1. \tag{7}$$

Then we multiply eq. (7) by t_i obtaining the $k \times k$ linear system

$$c_i + \sum_{j=0}^{k-1} t_i^T u_j c_j = -t_i^T HF(x_k) \qquad i = 0, \ldots, k-1, \tag{8}$$

in the unknowns $c_i = t_i^T z,\ i = 0, \ldots, k-1$. Hence z is obtained from (7) as

$$z = -HF(x_k) - \sum_{j=0}^{k-1} c_j u_j.$$

Finally, we compute by the CG an approximation s_k to $HAs = z$ with $s_0 = 0$ as the initial vector. Observe that, at each step, the coefficient matrix of system (8) is obtained simply by bordering the previous one, and the linear system $HAs = z$ is solved *concurrently* with the row-projection (Cimmino) method. It is straightforward to check that this system has a unique solution if and only if B_k is non singular for each k.

Note that one of the most expensive operations is represented by the construction of the preconditioner H. In the Quasi-Newton method this preprocessing stage is carried out once and for all since at each outer iteration we have to solve a linear system with the same $A = B_0$ as the coefficient matrix.

3 Convergence results

Now we give a convergence theorem under weak regularity conditions for F. Referring to system (6) for each $k = 0, 1, \ldots$, let us set

$$\varepsilon_k = \frac{\|s - s_k\|_{HA}}{\|s\|_{HA}}. \tag{9}$$

Our hypotheses are the following:
Assumptions 3.1. *Let $D \subset R^n$ be an open convex set. Let $F : D \to R^n$ be Frechet differentiable. Let $x^* \in D$ such that $F(x^*) = 0$ and $J(x^*)$ is non*

singular. Let us set

$$w(x,y) = \frac{\|F(x) - F(y) - J(x^*)(x-y)\|_2}{\|x-y\|_2}, \qquad \forall\ x, y \in D.$$

Let $S(r) = \{x : \|x^* - x\|_2\} \leq r\}$ *and let* $R > 0$ *such that* $S(r) \subset D$ *for* $0 < r \leq R$. *Then, for any* $0 < r \leq R$, *define*

$$\omega(r) = \sup_{(x,y)\in S(r)} w(x,y). \tag{10}$$

Accordingly, we assume that, for every $0 < \varepsilon < 1$, *it is* $\lim_{r\to 0} \sum_{k=0}^{\infty} \omega(\varepsilon^k r) = 0$.

Theorem 3.1. *Let Assumptions 3.1 hold. For every* $\varepsilon \in (0,1)$ *there exist a* $\delta > 0$ *and a* $\eta > 0$ *such that if* $\|x^* - x_0\|_{HA} \leq \delta$, $\|J(x^*) - A\|_2 \leq \eta$ *and* $\varepsilon_k \leq \varepsilon_0 < \varepsilon$ *for each* k, *then* B_k *is invertible, the sequence of iterates* $\{x_k\}$ *converges to* x^* *and*

$$\|x^* - x_{k+1}\|_{HA} \leq \varepsilon\ \|x^* - x_k\|_{HA} \tag{11}$$

Moreover if $\lim_{k\to\infty} \varepsilon_k = 0$, *the convergence is* q*-superlinear.*

The theorem states that the method converges, provided that the initial guesses x_0 and A are sufficiently good. On the other and, it also shows that a superlinear convergence can be achieved increasing step by step the accuracy by which one solves $HAs = z$. As for other iterative methods (see e.g. [4], [11], suitable modifications can be introduced in order to get global convergence.

The proof of the convergence being very technical, we defer it in Appendix A.

Table 1
Number m of rows of every block of the partitioning on each of the p nodes, numbers of nonzero elements n_1 – on nodes 1 and p – and n_2 – on the remaining nodes – respectively, referred to the Poisson and Bratu problems of size $n = 4096$.

p	m	l-blocks	n_1	n_2
1	4096	64	20224	–
2	2048	32	10112	–
4	1024	16	5024	5088
8	512	8	2480	2544
16	256	4	1208	1272
32	128	2	572	636

4 Computational aspects and numerical tests

The two procedures have been tested with the following nonlinear sparse problems $F(x) = (f_1, ..., f_n)^T = 0$:

(1) *Broyden tridiagonal problem* [2], with h a real parameter:

$$\begin{aligned} f_1(x) &= (3 - hx_1)x_1 - 2x_2 + 1 = 0, \\ f_i(x) &= -x_{i-1} + (3 - hx_i)x_i - 2x_{i+1} + 1 = 0, \quad i = 2, \ldots, n-1, \\ f_n(x) &= -x_{n-1} + (3 - hx_n)x_n + 1 = 0. \end{aligned}$$

(2) *Poisson problem,* obtained by application of the finite difference (FD) method to the two-dimensional boundary problem:

$$\Delta u = \frac{u^3}{1 + x^2 + y^2}, \qquad 0 \le x \le 1, \qquad 0 \le y \le 1,$$

$$\begin{aligned} u(0,y) &= 1, \qquad u(1,y) = 2 - e^y, \qquad 0 \le y \le 1, \\ u(x,0) &= 1, \qquad u(x,1) = 2 - e^x, \qquad 0 \le x \le 1, \end{aligned}$$

by discretizing the equation by the 5-point formula on a $l \times l$ grid with a total number of unknowns equals to $n = l \times l$. The resulting linear system is, as we know, a tridiagonal l-block system with every l-blocks of size l.

(3) *Bratu problem* [2] obtained by discretizing with FD in the unit square Ω the following two-dimensional boundary value problem:

$$-\Delta u - \lambda e^u = 0 \ \text{ in } \ \Omega, \qquad u = 0 \quad \text{on } \ \partial\Omega \tag{12}$$

for different values of the real parameter λ. It is known [10] that there exists a critical value of λ, λ^* such that problem (12) has two solutions for $\lambda < \lambda^*$ and no solutions for $\lambda > \lambda^*$.

To ensure a load balance between the processors, the Jacobian matrix is partitioned into p almost equal-sized blocks, each of them having $m = n/p$ rows. In the tridiagonal problem the numbers of nonzero elements assigned to each processor differ at most of 1, ($3n/p - 1$ for processors 1 and p, $3n/p$ for processor i, $i = 2, \ldots, p-1$). The remaining problems come from the discretization by finite differences of partial differential equations. The sparsity patterns of the matrices involved in these problems guarantee that again an almost equal number of nonzero elements is assigned to each processor. An example of the p blocks partitioning is described in Table 1 for the Poisson's and Bratu problems (with $l = 64$ and $n = 4096$), in which n_1 is the number of nonzero elements assigned at processor 1 and p and n_2 is the number of nonzero elements assigned to processor i, $i = 2, \ldots, p-1$. We notice that these two numbers are very close.

Table 2
Time (in seconds) and speedups obtained for solving the *tridiagonal* problem with $n = 131072$ with the *Inexact Newton-Cimmino method*, with $max_{NEWT} = 20$, $max_{CG} = 2$, $max_{LSQR} = 30$ and $\varepsilon_1 = 10^{-6}, \varepsilon_2 = 10^{-12}, \varepsilon_3 = 10^{-12}$. Time spent due to LSQR and Blas routines are also given (the latter is part of LSQR time).

$n = 131072$	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$
Time	126.2	79.4	58.6	47.5	42.2	40.0
speedup	1	1.6	2.2	2.7	3.0	3.2
k_{NEWT}	4	4	4	4	4	4
k_{CG}	2	2	2	2	2	2
k_{LSQR}	30	30	30	30	30	30
LSQR time	122.7	76.9	56.2	45.0	39.8	37.2
	97%	97%	96%	95%	94%	93%
Blas 1 time	57.3	42.1	36.0	31.9	30.3	29.5
Blas 2 time	59.6	29.0	14.5	7.5	3.8	1.9
Blas1 speedup	1	1.4	1.6	1.8	1.9	1.9
Blas2 speedup	1	2.0	4.1	7.9	15.7	31.4

The results are obtained on the CRAY T3E installed at CINECA (Bologna, Italy). The Fortran code runs under MPI implementation. We show the results obtained on a maximum of 32 processors.

The times shown in the forthcoming tables refer to the iterative part of the code: Newton outer iterations with updating of the Jacobian matrix (for Inexact Newton case only) and the right hand side vector. Each of this iteration essentially consists of a conjugate gradient procedure in which the pseudoresiduals $\delta_{i,k}$ are *concurrently* computed by k_{LSQR} LSQR iterations, within a tolerance ε_3. The sparse Jacobian matrix is stored by row, according to the *compressed sparse row* (CSR) format.

4.1 Results for the tridiagonal problem

The parameters used for this test problem are: $n = 131072$, $h = 2$, $x_0 = (-1, \ldots, -1)^T$. In order to compare correctly the speedups of the algorithm, we set $\varepsilon_2 = \varepsilon_3 = 10^{-12}$, thus forcing the Cimmino linear solver to stop at $k_{CG} = max_{CG} = 2$ inner iterations and the LSQR solver to stop at $k_{LSQR} = max_{LSQR} = 30$ iterations, for every p. With these parameters, the relative residual for the linear system has been $O(10^{-5})$ and the LSQR residual $O(10^{-10})$. Setting $\varepsilon_1 = 10^{-6}$ the tolerance for the nonlinear iteration, we found

Table 3
Time (in seconds) and speedups obtained for solving the *tridiagonal* problem with $n = 131072$ with the *Inexact Quasi Newton-Cimmino method,* with $max_{NEWT} = 20$, $max_{CG} = 2$, $max_{LSQR} = 30$ and $\varepsilon_1 = 10^{-6}, \varepsilon_2 = 10^{-12}, \varepsilon_3 = 10^{-12}$. Times spent due to LSQR and Blas routines are also given (the latter is part of LSQR time).

$n = 131072$	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$
Time	147.2	94.4	69.2	57.2	51.2	47.5
speedup	1	1.6	2.1	2.6	2.9	3.3
k_{NEWT}	5	5	5	5	5	5
k_{CG}	2	2	2	2	2	2
k_{LSQR}	30	30	30	30	30	30
LSQR time	142.6	90.3	65.2	52.7	47.3	43.2
	97%	96%	94%	92%	92%	91%
Blas1 time	66.6	49.2	41.4	37.3	36.0	35.2
Blas2 time	68.9	34.4	17.0	8.6	4.4	2.3
Blas1 speedup	1	1.4	1.6	1.8	1.9	1.9
Blas2 speedup	1	2.0	4.0	8.0	15.7	30.0

that an equal number $k_{NEWT} = 4$ (Newton) and $k_{NEWT} = 5$ (Quasi-Newton) outer iterations were sufficient to achieve convergence, for every p. We report the overall results of the Inexact Newton and Quasi-Newton methods in Tables 2 and 3, respectively.

From these Tables, where we also give the part of the LSQR time elapsed for calling the BLAS routines, we can see that the CPU time of the algorithm is essentially due to the LSQR solver. This will be confirmed by the other numerical experiments. Therefore we can conclude that the speedup values reported in the Tables are due to the sparse solver of the underdetermined systems of linear equations. The key to our procedure to be successful relies mainly on the scalability of this solver. From the tables we see that only the sparse *matvet* routine (BLAS2 kernel) gives the expected speedup, whereas BLAS1 kernels of length n require a number of operations $O(n)$ independent of the subproblem size (n/p).

From Tables 2 and 3 we can also see that the speedups obtained with the Inexact Quasi-Newton method are roughly the same as in the Inexact Newton method.

Since the parallelism of the whole algorithm essentially depends on the performance of the solver of the linearized system, at every nonlinear iteration, one single nonlinear iteration is sufficient to estimate the parallelism of the

Table 4
Time (in seconds) and speedups T_2/T_p obtained for solving the linear nonsymmetric Sameh problem of size $n = 4096$ with $\varepsilon_2 = 10^{-3}$ and $\varepsilon_3 = 10^{-12}$. Moreover, we give the average number of CG iterations k_{CG}, the inner LSQR k_{LSQR} iterations performed at each step and the overall number of LSQR iterations.

$n = 4096$	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$
Time	27.4	74.8	42.6	23.7	17.5	12.6
speedup	–	1	1.8	3.2	4.3	5.9
k_{CG}	1	28	45	64	90	127
k_{LSQR} (avg.)	2258	384	211	118	73	38
k_{LSQR} (tot.)	2258	10740	9481	7542	6569	4843
LSQR time	27.3	74.6	42.3	23.3	16.1	11.6
	99.9%	99.8%	99.7%	99.2%	96.3%	96.1%
Blas1 time	9.8	32.4	23.4	15.3	11.5	8.9
Blas2 time	16.3	37.6	14.8	4.9	2.0	0.7
Blas1 speedup		1	1.4	2.1	2.8	3.6
Blas2 speedup		1	2.5	7.6	18.8	53.7

whole algorithm. To this end, we report here the results of the block Cimmino method when applied to a *linear* test problem $Ax = b$, where A is the nonsymmetric sparse Sameh matrix. It is obtained by a 5-point finite difference discretization of the following PDE:

$$-u_{xx} - u_{yy} + 1000\, e^{xy}\, (u_x - u_y) = g, \qquad u = x + y,$$

on a 64×64 grid. The symmetric part of A is not positive definite. It is known (see [12], [15]) that restarted Krylov subspace methods stagnate or fail to converge with this problem. Instead, convergence is attained by the Cimmino method. The right hand side is chosen so that the solution is $x = (1, 2, \ldots, n)^T$. Starting from a zero initial guess, we stop the iterations as soon as the relative residual norm $\|r_m\|/\|r_0\|$ is smaller than a prescribed tolerance ε_2. Taking advantage of the experiments on the convergence of the two nonlinear problems, we could set the tolerance ε_2 to a not too small accuracy ($\varepsilon_2 = 10^{-3}$ in our test). On the contrary, since the LSQR routine has to solve accurately the underdetermined subproblems, we set $\varepsilon_3 = 10^{-12}$.

The results are shown in Table 4. From this Table it is worth noting that with $p = 1$ the Cimmino method acts as an *exact* preconditioner applied to $n \times n$ problem and hence the solution is attained at the very first iteration (with $k_{LSQR} = 2258$ LSQR iterations). With $p \geq 2$, the number of Cimmino iterations and the corresponding CPU time increase largely, due to the

Table 5
Time (in seconds) and speedups T_2/T_p obtained for solving the Poisson problem of size $n = 4096$ with the Inexact Newton method with $\varepsilon_1 = 10^{-3}, \varepsilon_2 = 10^{-4}$ and $\varepsilon_3 = 10^{-12}$. Times spent due to LSQR and Blas routines are also given (the latter is part of LSQR time). Moreover, we give the number of outer iterations k_{NEWT}, the average number of CG iterations k_{CG}, the inner LSQR k_{LSQR} iterations performed at each step and the overall number of LSQR iterations.

$n = 4096$	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$
Time	201.6	2120.7	1521.8	702.1	311.8	159.8
speedup		1	1.4	3.0	6.8	13.3
k_{NEWT}	2	2	2	2	2	2
k_{CG}	1	89	212	303	391	525
k_{LSQR} (avg.)	4276	1715	813	372	161	68
k_{LSQR} (tot.)	4276	165400	176960	113920	62841	35428

ill-conditioning of the Laplace equation. As expected, we will see that this also occurs with the Poisson and Bratu test problems. For these problems, it is therefore reasonable to evaluate the performance of the parallel Cimmino method with respect to the T_2 CPU time ($p = 2$). The speedups reported in Table 4 are consistently computed as $S_p = T_2/T_p$, $p \geq 2$. The promising speedup values shown in the table account for the fact that both the numbers of LSQR iterations (k_{LSQR}) and the CPU time which they require, decrease with growing p, opposed to an increasing number of CG iterations (k_{CG}) required to solve the preconditioned systems $HAx = Hb$. This behavior is summarized in Table 4 by the number of LSQR iterations which are shown to decrease, starting from the $p = 2$ case on. In Table 4, we finally note that starting from 8 processors we obtain a total CPU time which is less than the $p = 1$ CPU time.

4.2 Results for the Poisson problem

For comparison with the linear test problem, we restricted ourselves to test a problem of size $n = 4096$, relative to a grid of $n = l \times l$ nodes, $l = 64$, see Table 1. The test parameters were $\varepsilon_1 = 10^{-3}$, $\varepsilon_2 = 10^{-4}$, $\varepsilon_3 = 10^{-12}$, starting from an initial guess $x_0 = (-1, \ldots, -1)^T$. We see that only two outer iterations were required by both methods to achieve the exit test. Therefore the two methods took almost the same time and an equal number of iterations. The Inexact Quasi-Newton method should be tested with more difficult problems with respect to the evaluation of the Jacobian matrix, in order to show computational advantages with respect to the Inexact Newton approach. The results are summarized in Table 5 which shows the high speedup values for

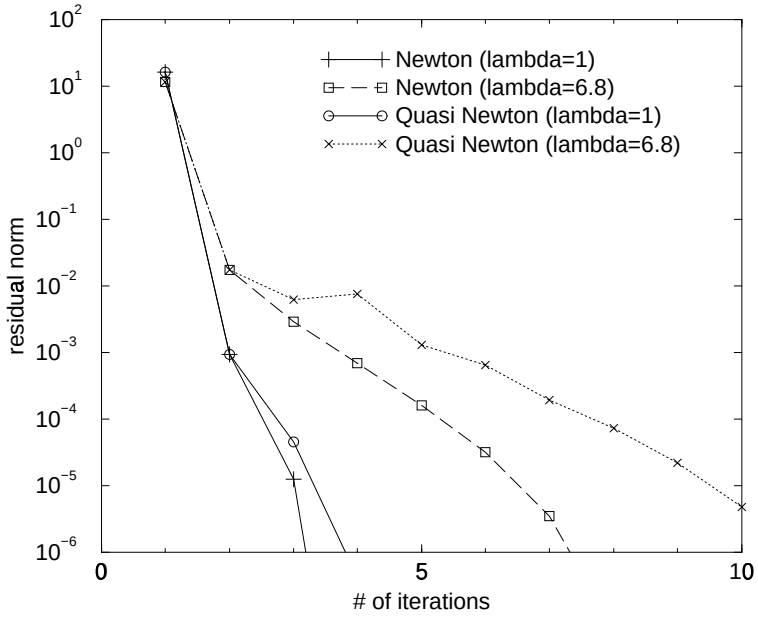

Fig. 1. Convergence profile of the non linear procedures in the solution of the Bratu problem with $\lambda = 1$ and $\lambda = 6.8$ (λ-values reported between parentheses).

a relatively small problem. We may note that the total LSQR iterations decrease, from $p = 4$ to $p = 32$ by a factor 5, proving that the condition number of the underdetermined problems dramatically decrease with their size.

4.3 Results for the Bratu problem

We report in Figure 1 the convergence profiles of the two nonlinear procedures applied to the test problem with $\lambda = 1$ (weak nonlinearity) and $\lambda = 6.8$ (near the critical point). In the $\lambda = 1$ case the two profiles are very similar, showing that the two methods behave in the same way (they both achieve the exit test after 4 iterations). In the $\lambda = 6.8$ case, the Newton method converges after 7 iterations while 10 iterations are required by the Quasi-Newton method. Table 6 shows the results obtained with $\lambda = 1$ of the Newton case only, on a 64×64 mesh with $n = 4096$. We may notice again that the total number of LSQR iteration dramatically increase from $p = 1$ to $p = 2$ processors, but it decreases from $p = 4$ processors. Therefore, as already pointed out in the linear case, the speedups $T_2/T_p, p > 2$ are larger than the speedups exhibited in the tridiagonal case.

Table 6
The same as Table 5 – for the Bratu problem – with $n = 4096$ and $\lambda = 1.0$ and with $\varepsilon_1 = 10^{-4}, \varepsilon_2 = 10^{-5}, \varepsilon_3 = 10^{-12}$.

$n = 4096$	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$
Time	93.36	1029.51	636.207	387.12	188.77	118.60
speedup	–	1	1.6	2.7	5.5	8.7
k_{NEWT}	4	4	4	4	4	4
k_{CG}	1	48	86	141	230	318
k_{LSQR} (avg.)	1300	1021	341	278	119	62
k_{LSQR} (tot.)	1300	49000	47888	40336	27512	19676

5 Row-orthogonal partitioning

As already pointed out in the Introduction, to overcome the problem of the costly solution of the least square subproblems, we propose a suitable block row partitioning of the matrix A in such a way that $A_i A_i^T = I, \quad i = 1, \ldots q$, and consequently, $A_i^+ = A_i^T$. This allows to simplify the solution of the least squares subproblems at each step of the inner iteration. This partitioning [15] is always possible for every sparse matrix and produces a number q of blocks A_i whose rows are mutually orthogonal. As an example, we show in Figure 2 the pattern of the Sameh linear test problem after the reordering in 7 row-

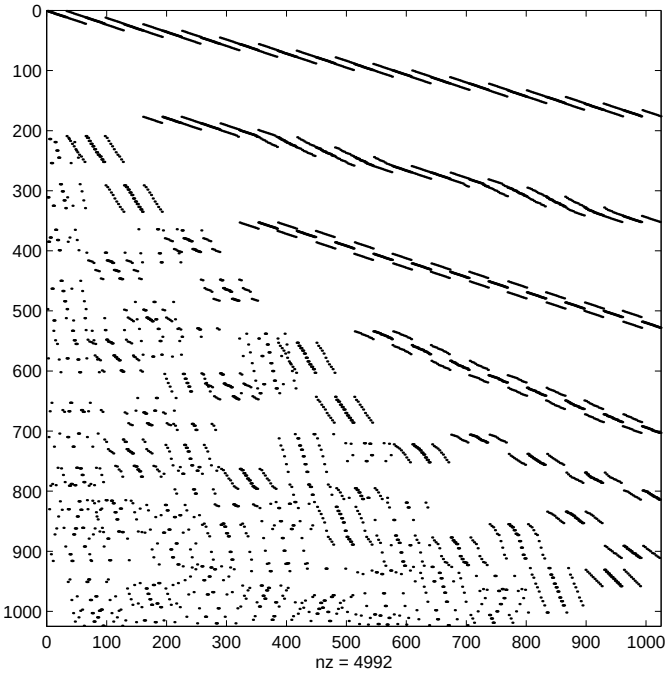

Fig. 2. Pattern of the Sameh linear test problem after the reordering in 7 row-orthogonal blocks

Table 7
Time (in seconds), speedups T_1/T_p, number of outer (k_{NEWT}) and inner iterations (k_{CG}) obtained for solving the three test problems, using the row-orthogonal partitioning.

Poisson problem, $\varepsilon_1 = 10^{-4}, \varepsilon_2 = 10^{-5}$.					
$n = 4096$	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$
Time (speedup)	21.66	17.22 (1.3)	17.37 (1.3)	18.15 (1.2)	20.49 (1.1)
k_{NEWT}	2	2	2	2	2
k_{CG}	1123	1123	1123	1123	1123

Bratu problem, $\varepsilon_1 = 10^{-4}, \varepsilon_2 = 10^{-5}$.					
$n = 4096$	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$
Time (speedup)	13.40	10.22 (1.3)	9.92 (1.4)	10.66 (1.3)	16.02 (0.8)
k_{NEWT}	4	4	4	4	4
k_{CG}	630	630	630	630	630

Linear problem, $\varepsilon_2 = 10^{-8}$					
$n = 4096$	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$
Time (speedup)	6.32	4.60 (1.4)	4.00 (1.6)	4.08 (1.6)	4.60 (1.4)
k_{CG}	696	695	694	694	694

orthogonal blocks. The numerical results relative to the row-partitioning for some of the test problems of the previous section are shown in Table 7. From the Tables 4, 5, 6, and 7 we can make the following observations.

The Conjugate Gradient applied to the row-orthogonal partitioned matrices takes a larger number of iterations with respect to the LSQR case. For example, in the Poisson test case the number of outer iterations are constantly equal to 1123 (see Table 7) in the orthogonal case while they range from 1 to 525 (see Table 5) using the LSQR algorithm. The same behavior is evidenced by the other test problems.

On the contrary, the orthogonal variant of the algorithm is more convenient from the point of view of CPU time, as can be seen by comparing the overall CPU times in the three test cases for a fixed number of processors. Note that now a single iteration is much less costly than in the LSQR algorithm, since the solution of a linear subproblem is replaced by a cheap multiplication for a diagonal matrix (or even the identity). Moreover, the speedups in the second algorithm are correctly computed as T_1/T_p because in this case the CPU

Table 8
CPU time needed by the matrix vector products and by the reduce operation in the linear test case.

p	$w = A_i v$	(T_p/T_1)	$w = A_i^T v$	(T_p/T_1)	mpi_allreduce (%)	overall
1	1.40	(-)	1.60	(-)	0.36 (06)	6.32
2	0.70	(2.0)	0.86	(1.8)	0.92 (20)	4.60
4	0.38	(3.7)	0.54	(3.0)	1.50 (25)	4.00
8	0.20	(7.0)	0.36	(4.4)	2.08 (51)	4.08
16	0.10	(13.9)	0.24	(6.7)	2.66 (58)	4.60

time decreases from 1 processor forward. However, they are not completely satisfactory, reaching the maximum value of 1.4 for $p = 4$ processors. This fact is mainly due to the cost of the communication routine MPI_ALLREDUCE which performs the communication of the local pseudoresiduals and their sums on every processor. This operation is costly, and its cost increases with the number of processors. Table 8 refers to the linear problem, and yields the time spent by the most expensive routine (the matrix-vector product) and by the MPI_ALLREDUCE routine. Note the good speedup obtained by the matrix-vector product, while the time spent by the MPI_ALLREDUCE routine becomes a large part of the overall CPU time as the number of processors increase, going to more than 50% in the case with 16 processors. Differently, in the LSQR case the times spent by the communication represent a little portion compared to the much higher times required to solve the least squares subproblems.

6 Conclusions and future topics

In this paper we have proposed an Inexact Newton and Quasi Newton method, whose linearized system is solved by a row-projection iterative scheme (Cimmino method). In principle, this approach is well suited for a parallel environment since it requires independent solutions of p underdetermined linear subproblems, p being the number of processors.

We have first proved that, under suitable hypotheses, the Inexact Quasi-Newton Cimmino method achieves at least linear convergence in the energy norm induced by the preconditioned matrix HA, A being an approximation of the Jacobian matrix. The numerical experiments on the CRAY T3E on problems of large size point out that the solution of the underdetermined subproblems represent the major part of the computation. The speedups obtained (test problems 1) are not fully satisfactory, due to the poor scalability of BLAS1 routines called by LSQR solver.

Moreover, the number of Cimmino iterations may increase with the number of processors (test problems 2 and 3), corresponding to a worsening of the action of the preconditioner. In the $p = 1$ case, the Cimmino method converges in only one iteration, being $HA = I$. At the same time the subproblems become better conditioned with increasing p, and therefore they require less iterations to be solved. The balance between these two aspects result in a decrease of the overall number of inner iterations and consequent high speedup values $T_p/T_2, p \geq 2$.

Some preliminary results show that the algorithm with the row orthogonal block partitioning of the Jacobian matrix, despite of a larger number of outer iterations with respect to the LSQR case is more convenient from the point of view of CPU time. Moreover, the CPU time correctly decreases from 1 processor forward, and the speedups are correctly evaluated as $S_p = T_1/T_p,\ p \geq 1$. They are however not yet satisfactory. This fact is mainly due to the cost of the communication routine MPI_ALLREDUCE which performs the communication of the local pseudoresiduals and their sums on every processor. This operation is costly, and its cost increases with the number of processors.

Finally, the advantage of using the Quasi-Newton method instead of the Newton method relies on the fact that only the initial approximation A of the Jacobian matrix is required, without the updating of the Jacobian matrix at every nonlinear step. This advantage will be more evident with other test problems in which the Jacobian matrix is not available or difficult to compute and requiring a higher number of iterations.

A Proof of Theorem 3.1

To prove the theorem 3.1, we need the following

Lemma A.1. *Let $J(x^*)$ be invertible, then there are positive constants η_0, σ_1 and σ_2 such that for every $n \times n$ matrix A with $\|J(x^*) - A\|_2 \leq \eta_0$ it is*

$$\sigma_1^2 \langle x, x \rangle \leq \langle HAx, x \rangle \leq \sigma_2^2 \langle x, x \rangle$$

Proof of Lemma A.1. Let us set $A^* = J(x^*)$ and A^* be partitioned into p block rows $A_i, i = 1, \ldots, p$, i.e. $A^{*^T} = \left[A_1^{*^T}, A_2^{*^T}, \ldots, A_p^{*^T}\right]$. Accordingly, let us set

$$H^* = [A_1^{*^+}, \ldots, A_i^{*^+}, \ldots, A_p^{*^+}].$$

Then

$$H^* A^* = P_1^* + \cdots + P_i^* + \cdots + P_p^*$$

where $P_i^* = A_i^{*^+} A_i^*$. Clearly, there are positive constants ρ_1 and ρ_2 such

$$\rho_1 \|x\|_2^2 \leq \|x\|_{H^*A}^2 \leq \rho_2 \|x\|_2^2.$$

Let A and η_0 such that if $\|J(x^*) - A\| \leq \eta_0$, then A is invertible. Hence, partitioning A as A^*, it follows that for each $i = 1, \ldots, p$,

$$\operatorname{rank}(A_i) = \operatorname{rank}(A_i^*).$$

As well known (see [14]) this ensures that, if η_0 is sufficiently small, then there is a constant c, independent of A, such that

$$\|P_i^* - P_i\|_2 \leq c\, \eta_0,$$

for every $i = 1, \ldots, p$, so that we have

$$\|H^* A^* - HA\|_2 \leq pc\eta_0.$$

Then, let η_0 be that $pc\eta_0 < \rho_1$. We easily get

$$(\rho_1 - pc\eta_0)\|x\|_2^2 \leq \|x\|_{HA}^2.$$

Obviously

$$\|x\|_{HA}^2 \leq (\rho_2 + pc\eta_0)\|x\|_2^2.$$

□

Proof of Theorem 3.1. Referring to the notation of the previous Lemma A.1, let us set

$$\beta^* = \sigma_2 \, \|J^{-1}(x^*)\|_2.$$

Let us take $\eta \leq \eta_0$ and δ such that the following conditions are fulfilled:

$$2\beta^* \frac{\eta}{\sigma_1} < 1 \tag{A.1}$$

$$r = \sigma_2 \delta \leq R \tag{A.2}$$

$$\sum_{k=0}^{\infty} \omega\left(\varepsilon^k r\right) \leq \eta \tag{A.3}$$

$$(1 + \varepsilon_0)\, \beta \left(\frac{2\eta}{\sigma_1} + \frac{\omega(r)}{\sigma_1} \right) + \varepsilon_0 \leq \varepsilon, \text{where } \beta = \frac{\beta^*}{1 - 2\beta^* \dfrac{\eta}{\sigma_1}} < 1 \tag{A.4}$$

By Lemma A.1, if x_0 is such that $\|x^* - x_0\|_2 \leq \delta$ then $\|x^* - x_0\|_{HA} \leq r$. Setting $C_0 = A$ we have $B_0 = HC_0$ and

$$\|(C_0 - J(x^*))\,(HA)^{-1/2}\|_2 \leq \frac{\eta}{\sigma_1} \tag{A.5}$$

using the hypothesis and Lemma A.1 again. Therefore we have

$$\|(HA)^{1/2}C_0^{-1}\|_2 \leq \beta, \tag{A.6}$$

using the classic inequality

$$\|(S+T)^{-1}\| \leq \frac{\|S^{-1}\|}{1-\|S^{-1}\|\|T\|}$$

with S and T two matrices. We now assume that, for $j = 0, 1, \dots, k$,

$$\|(C_j - J(x^*))\,(HA)^{-1/2}\|_2 \leq 2\frac{\eta}{\sigma_1}, \tag{A.7}$$

where $B_j = HC_j$. The proof, by induction, is given below; for $j = 0$ see (A.5). Thus C_j is invertible and, with the same argument used to prove (A.6), one can obtain that

$$\|(HA)^{1/2}C_j^{-1}\|_2 \leq \beta. \tag{A.8}$$

Now we can prove (11) by induction. First of all we have

$$\|x^* - x_{k+1}\|_{HA} \leq \|x^* - x_k - s\|_{HA} + \varepsilon_k\|s\|_{HA}.$$

Then, since

$$\|s\|_{HA} \leq \|x^* - x_k - s\|_{HA} + \|x^* - x_k\|_{HA}$$

we have

$$\|x^* - x_{k+1}\|_{HA} \leq (1+\varepsilon_0)\|x^* - x_k - s\|_{HA} + \varepsilon_0\|x^* - x_k\|_{HA}. \tag{A.9}$$

Moreover (adding and subtracting $-J(x^*)$) we get

$$\begin{aligned}(HA)^{1/2}\,(x^* - x_k - s) &= \left[(HA)^{1/2}C_k^{-1}\right]\,[C_k\,(x^* - x_k) + F(x_k)] = \\ &= \left[(HA)^{1/2}C_k^{-1}\right] \times \left\{[(C_k - J(x^*))(HA)^{-1/2}(HA)^{1/2}\,(x^* - x_k)] - \right. \\ &\qquad \left. - \,[(F(x^*) - F(x_k) - J(x^*)\,(x^* - x_k))]\right\}.\end{aligned}$$

Hence, taking (HA)-norms, by (A.8), (A.7) and by Assumptions 3.1, we obtain

$$\begin{aligned}&\|x^* - x_k - s\|_{HA} \leq \beta\left(\frac{2\eta}{\sigma_1}\|x^* - x_k\|_{HA} + \omega(\varepsilon^k r)\|x^* - x_k\|_2\right) \leq \\ &\leq \beta\left(\frac{2\eta}{\sigma_1} + \frac{\omega(\varepsilon^k r)}{\sigma_1}\right)\|x^* - x_k\|_{HA},\end{aligned}$$

and finally

$$\|x^* - x_{k+1}\|_{HA} \leq \varepsilon \ \|x^* - x_k\|_{HA} \leq \varepsilon^{k+1}\|x^* - x_0\|_{HA} \leq \ldots \leq \varepsilon^{k+1} r$$

using (A.4) and (A.9).

We now prove (A.7). Using the Broyden update, note that, for $j = 0, 1, \ldots, k+1$ we have

$$C_{j+1} = C_j + \frac{F(x_{j+1}) - F(x_j) - C_j s_j}{\|s_j\|_{HA}} \ \frac{(HAs_j)^T}{\|s_j\|_{HA}} \qquad k = 0, 1, \ldots \ .$$

Then we have

$$C_{j+1}(HA)^{-1/2} = C_j(HA)^{-1/2} + \frac{(F(x_{j+1}) - F(x_j) - C_j s_j)\,((HA)^{1/2}s_j)^T}{\|s_j\|^2_{HA}},$$

so that

$$\begin{aligned}
&(C_{j+1} - J(x^*))\,(HA)^{-1/2} = \\
&\quad = (C_j - J(x^*))\,(HA)^{-1/2}\left(I - \frac{(HA)^{1/2}s_j((HA)^{1/2}s_j)^T)}{\|s_j\|^2_{HA}}\right) + \\
&\quad + \frac{(F(x_{j+1}) - F(x_j) - J(x^*)s_j)\,((HA)^{1/2}s_j)^T}{\|s_j\|^2_{HA}}.
\end{aligned}$$

Taking 2-norms, by (10) and the convergence hypotheses and by Lemma A.1, we get

$$\begin{aligned}
&\|(C_{j+1} - J(x^*))(HA)^{-1/2}\|_2 \leq \|(C_j - J(x^*))(HA)^{-1/2}\|_2 \ + \\
&+ \ \frac{\|F(x_{j+1}) - F(x_j) - J(x^*)s_j\|_2}{\|s_j\|_{HA}} \ \frac{\|(HA)^{1/2}s_j)\|_2}{\|s_j\|_{HA}} \ \leq \\
&\leq \ \|(C_j - J(x^*))(HA)^{-1/2}\|_2 + \omega(\varepsilon^j r)\frac{1}{\sigma_1}. \qquad \text{(A.10)}
\end{aligned}$$

Hence, recursively we have

$$\begin{aligned}
&\|(C_{k+1} - J(x^*))\,(HA)^{-1/2}\|_2 \leq \|(C_k - J(x^*))\,(HA)^{-1/2}\|_2 + \omega(\varepsilon^k r)\frac{1}{\sigma_1} \leq \\
&\ldots \leq \|(C_0 - J(x^*))(HA)^{-1/2}\|_2 + \frac{1}{\sigma_1}\sum_{j=0}^{k}\omega(\varepsilon^j r)\ \frac{2\eta}{\sigma_1},
\end{aligned}$$

using (A.5) and (A.3).

In order to show the q-superlinear convergence when $\lim_{k\to\infty}\varepsilon_k = 0$, in the space of the $n \times n$ matrices let us introduce the scalar product $\langle M, N\rangle_{HA} = \sum_{j=1}^{n}\langle Me_j, Ne_j\rangle_{HA}$, where $\{e_j\}$ is some system that is orthogonal with respect to the scalar product $\langle\ ,\ \rangle_{HA}$ in R^n. Then, considering the norm $|||M|||_{HA} = \sqrt{\langle M, M\rangle_{HA}}$, the following lemma can be easily proved.

Lemma A.2. *For every matrix M and for every $z \in R^n$ we have*

$$|||M(I - \frac{z(HAz)^T}{\|z\|^2_{HA}})|||^2_{HA} = |||M|||^2_{HA} - \frac{\|Mz\|^2_{HA}}{\|z\|^2_{HA}}.$$

Using this and assuming that $\lim_{k\to\infty}\varepsilon_k = 0$, one proves the q-superlinear convergence by some standard arguments like those used in Th. 8.2.2 of [8]. □

Acknowledgements

Work supported in part by the MURST funds. The first author has also been supported by EC contract # IC15 CT96-211.

References

[1] Arioli, M., Duff, I., Noailles, J., and Ruiz, D., A block projection method for sparse matrices, *SIAM J. Sci. Stat. Comput.* **13 (1)**, 47-70, 1992.

[2] Bongartz, I., Conn, A.R., Gould, N.I.M., and Toint, P.L., CUTE: *Constrained and unconstrained testing environment*, Research Report, IBM T.J. Watson Research Center, Yorktown Heights, NY, 1993.

[3] Bramley R., and Sameh, A., Row projections methods for large nonsymmetric linear systems, *SIAM J. Sci. Stat. Comput.* **13 (1)**, 168-193, 1992.

[4] Brown P.N. and Saad Y., Hybrid Krylov methods for nonlinear systems of equations, *SIAM J. Sci. Stat. Comput.* **11**, 450-481, 1990.

[5] Cimmino, G., Calcolo approssimato per le Soluzioni di Equazioni lineari, *La Ricerca Scientifica*, 1, 326-333, 1938

[6] Dembo, R.S. and Steihaug, T., Truncated Newton algorithms for large-scale unconstrained optimization, *Series # 48, Yale University, New Haven, CT, September 1980.*

[7] Dembo, R.S., Eisenstat, S.C., and Steihaug, T., Inexact Newton methods, *SIAM. J. Numer. Anal.* **19**, 400-408, 1982.

[8] Dennis, J.E. Jr., and Schnabel, S., Numerical Methods for Unconstrained Optimization and Nonlinear Equations, *SIAM, Philadelphia, PA, 1996.*

[9] Eisenstat, S.C., and Walker, H.F., Choosing the forcing terms in an Inexact Newton method, *SIAM. J. Sci Comput.* **17-1**, 16-32, 1996.

[10] Fokkema D. R., Slejipen G.L.G. and Van der Vorst H.A., Accelerated Inexact Newton schemes for large systems of nonlinear equations. *SIAM J. Sci. Comput.*, 19 (2), 657-674, 1997.

[11] Griewank A., The local convergence of Broyden like-methods on Lipschitzians problems in Hilbert spaces, *SIAM J. Numer. Anal.* **24**, 684-705, 1987.

[12] Kamath, C. and Sameh, A., A projection method for solving nonsymmetric linear systems on multiprocessors, *Parallel Computing* **9**, 291-312, 1988/89.

[13] Paige, G.G. and Saunders, M.A., LSQR: An algorithm for sparse linear equations and sparse least squares, *ACM Trans. Math. Software* **8**, 43-71, 1982.

[14] Stewart, G.W., On the perturbation of pseudo-inverses, projections and linear least squares problems, *SIAM Review* **19**, 634-662, 1977.

[15] Zilli, G., Parallel implementation of a row-projection method for solving sparse linear systems, *Supercomputer* **53, X-1**, 33-43, 1993.

[16] Zilli, G., Parallel method for sparse non-symmetric linear and non-linear systems of equations on a transputer network, *Supercomputer* **6, XII-4**, 4-15, 1996.

[17] Zilli, G. and Bergamaschi, L., Truncated block Newton and Quasi Newton methods for sparse systems of nonlinear equations. Experiments on parallel platforms in: *M. Bubak, J. Dongarra, J. Wasniewski (Eds.), Recent advances in PVM and MPI, Lectures Notes in Computer Sciences,* 1332, Springer, 390-397, 1997.